Fibrational semantics for many-valued logic programs: grounds for non-groundness.

Ekaterina Komendantskaya[1] and John Power[2]

[1] INRIA Sophia Antipolis, France
ekaterina.komendantskaya@inria.fr
[2] University of Bath, UK
A.J.Power@bath.ac.uk *

Abstract. We introduce a fibrational semantics for many-valued logic programming, use it to define an SLD-resolution for annotation-free many valued logic programs as defined by Fitting, and prove a soundness and completeness result relating the two. We show that fibrational semantics corresponds with the traditional declarative (ground) semantics and deduce a soundness and completeness result for our SLD-resolution algorithm with respect to the ground semantics.
Key words: Many-valued logic programs, categorical logic, fibrational semantics, ground semantics, SLD-resolution.

1 Introduction

Declarative semantics for logic programming characterises logic programs from the model-theoretic point of view, in particular, it shows a procedure for computing (Herbrand) models of logic programs. Commonly, it is given by defining an appropriate semantic operator that works recursively over the *Herbrand base* and the ground instances of clauses and finally settles on the least Herbrand model of a program, [12]. An assortment of many-valued logic programs has received appropriate declarative semantics: *annotation-free logic programs* [6,7,3,16], *implication-based logic programs* [17], *annotated logic programs* [1,4,8,13]. The declarative semantics received algebraic [4] and categorical [5] account.

Another type of semantics for logic programming is called *operational*. Operational semantics gives a proof-theoretic view on logic programming. Often, it is given by the SLD-resolution, [12]. As for many-valued generalisations of logic programming, the (SLD) resolution procedures were suggested for a number of different many-valued logic programs, [1,8,13,17,3,16].

A third type of semantics, a *fibrational semantics* for logic programming was suggested; [9]. It gave structural (categorical) characterisation of the syntax of logic programs. Unlike declarative semantics, fibrational semantics does not use Herbrand models. As a consequence, this kind of semantics does not

* The authors thank the grant "Categorical Semantics for Natural Models of Computation" by the Royal Society/Royal Irish Academy.

depend on ground instances of terms, atoms and clauses. Instead, fibrational semantics shows that the syntax of a logic program - sorts of variables, arities of terms, arities of conjunctions in the clause bodies and "$\leftarrow$" , - induces a particular structure that characterises the logic program uniquely up to the variable renaming. We will explain this in Section 3. Due to its non-groundness, fibrational semantics can be easily and naturally related to operational semantics and SLD-resolution: neither fibrational, nor operational semantics depend on ground instances of atoms. This is why, the fibrational semantics was used to give a category-theoretic account of SLD-resolution [9,15].

Despite of its elegance, the fibrational semantics has never been extended to any kind of non-classical logic programming. And there was a serious obstacle for such extensions: namely, the fibrational semantics of [9] gave no answer to the question of what role a truth value assignment plays in the new semantics. In fact, this question had no particular importance in case of classical, two valued, logic programs that were analysed in [9], because the evaluation *true* could be automatically assumed for all the clauses constituting a program. And thus, without explicitly mentioning, the fibrational semantics [9] structurally interpreted true unit clauses, and true logical implications between clause bodies and clause heads.

However, in case of many-valued extensions, one cannot simply assume that all the unit clauses are true. Moreover, in case if truth values are not allowed as annotations [6,7], one cannot deduce the truth value of a formula looking simply at the structure of a logic program. Furthermore, it is impossible to assign a truth value to a non-ground formula. In this paper, we analyse this situation categorically. In Section 4 we give a *ground semantics* to many-valued logic programs, respecting the tradition [6,7,4,1] to assign truth values only to ground formulae. In Section 5 we give a fibrational semantics to annotation-free logic programs, and prove that it is equivalent to the ground semantics.

We believe that Proposition 1 and Theorem 1 establishing precise relations between ground and fibrational semantics give theoretical justification for fibrational semantics and break new grounds for future development of the fibrational approach to non-classical logic programming. As an evidence that fibrational semantics can lead to useful applications, we show, in Section 6, that the fibrational semantics for many-valued logic programs gives rise to a novel algorithm of SLD-resolution for annotation-free logic programs. We prove its soundness and completeness relative to the ground and fibrational semantics of Sections 4, 5. In comparison to alternative approaches to many-valued resolution algorithms in [3,16], this novel algorithm provides the ideal compromise between expressiveness and efficiency, as we briefly explain in Section 7.

2 Many-Valued Logic Programs

A conventional (two-valued) logic program [12] consists of a finite set of clauses, some of which form its core, and the rest of which form a database.

Example 1. Let GC (for graph connectivity) denote the logic program with core $(\text{connected}(x, x) \leftarrow)$, $(\text{connected}(x, y) \leftarrow \text{edge}(x, z), \text{connected}(z, y))$.

A database for GC lists the edges of a particular graph: $\text{edge}(a, b) \leftarrow$, $\text{edge}(b, c) \leftarrow$, ...

For the formal analysis of this paper, we need more precision, as follows.

Definition 1. *Given a set $\mathcal{T}$, the set Sort($\mathcal{T}$) of* sorts *generated by $\mathcal{T}$ is the set of all finite, possibly empty, sequences of elements of $\mathcal{T}$.*

We use T_1, T_2 etc., to refer to elements of $\mathcal{T}$; and $\overline{T} = T_1, \ldots, T_n$ to refer to sequences of elements of Sort($\mathcal{T}$). Using categorical notation, we will use the symbol 1 to denote the terminal object (given the empty sequence) in a Cartesian category Sort($\mathcal{T}$), where sequences $T_1, \ldots, T_n$ are seen as finite products. More generally, we will use symbol 1 throughout the paper whenever we talk about an empty product in a given Cartesian category.

Definition 2. *A* sorted language *is a triple $\mathcal{L} = (\mathcal{T}, \mathcal{F}, \mathcal{P})$ consisting of*

- *a set $\mathcal{T}$ of primitive sorts;*
- *for each $\overline{T} \in$ Sort($\mathcal{T}$) and a primitive sort $T \in \mathcal{T}$, a set $\mathcal{F}(\overline{T}, T)$ of* function symbols *of sort $(\overline{T}, T)$, and*
- *for each $\overline{T} \in$ Sort($\mathcal{T}$), a set $\mathcal{P}(\overline{T})$ of* predicate symbols *of sort $\overline{T}$.*

Given a sorted language $\mathcal{L} = (\mathcal{T}, \mathcal{F}, \mathcal{P})$ and a set V of variables, we can define terms and atomic formulae as usual, all of these with sorts.

Example 2. The language underlying the logic program from Example 1 is a triple $(\mathcal{T}, \mathcal{F}, \mathcal{P})$ as follows: $\mathcal{T} = \{D\}$; $\mathcal{F}(1, D) = \{a, b, c, \ldots\}$, otherwise $\mathcal{F}(\overline{T}, T)$ is empty; and $\mathcal{P}(DD) = \{\text{connected}, \text{edge}\}$, otherwise $\mathcal{P}(\overline{T})$ is empty.

So, there is one sort D. And there are several nullary function symbols, i.e, constants $a, b, c \ldots$. And there are two binary predicates "connected" and "edge". The sortedness of the predicate amounts simply to their being binary, as the language is single sorted.

Example 3. Suppose we wish to enumerate edges of a given graph using the set of natural numbers. This would require the use of the second sort N. We use predicate "rank" for this purpose. E.g, the clause $(\text{rank}(0, a, b) \leftarrow \text{edge}(a, b))$ describes the basic step of enumeration. Then, we redefine $\mathcal{T}$ of Example 2: $\mathcal{T} = \{D, N\}$; and add $\mathcal{F}(1, N) = \{0, 1, 2, 3, \ldots\}$; and $\mathcal{P}(NDD) = \{\text{rank}\}$. One can use standard predicates "odd" and "even" over natural numbers. Then we would additionally have $\mathcal{P}(N) = \{\text{odd}, \text{even}\}$.

Definition 3. *A* sorted logic program *Γ over the language $\mathcal{L}$ consists of a finite set of clauses $(\overline{T}, \overline{\varphi}, \varphi)$, where $\overline{T}$ is a* sort *of a clause, $\overline{\varphi}$ is a formula of the form $P_1(\overline{t_1}) \wedge \ldots \wedge P_n(\overline{t_n})$ and φ is an atomic formula of the form $P(\overline{t})$; both $\overline{\varphi}$ and φ are of sort $\overline{T}$.*

Example 4. Example 1 is an example of a logic program with one sort. In Example 2, we expressed the language formally, with the sort denoted by D. The logic program has two clauses $(\text{connected}(x, x) \leftarrow)$ and $(\text{connected}(x, y) \leftarrow \text{edge}(x, z), \text{connected}(z, y))$ in its core. They are of sorts D and DDD respectively. Additional clauses $(\text{edge}(a, b) \leftarrow)$, $(\text{edge}(b, c) \leftarrow)$ are of sort 1.

Thus, the sort of a clause depends on the number and sorts of free variables. That is, although the predicate "connected" is binary, the clause $(\text{connected}(a, x) \leftarrow)$ would be of sort D. The clause $(\text{rank}(0, a, b) \leftarrow \text{edge}(a, b))$ from Example 3 would be of sort 1. The clause $(\text{rank}(n + 1, x, y) \leftarrow \text{edge}(x, y), \text{rank}(n, z, x))$ would be of sort $NDDD$.

Many-valued annotation-free logic programs [6,7], are formally the same as two-valued logic programs, see Definition 3. But while each atomic ground formula of a two-valued logic program is given an interpretation in $\{0, 1\}$, an atomic formula of a many-valued logic program receives an interpretation in an arbitrary specified preorder Ω with finite meets.

Example 5. Our leading example of an annotation-free logic program is as follows. Let Ω be the unit interval $[0, 1]$. The logic program of Example 1 is, by definition, also an annotation-free ($[0, 1]$-based) logic program. But each ground atom, e.g., $(\text{edge}(a, b))$ or $(\text{connected}(a, b))$, is assigned a truth value from $[0, 1]$, (cf. the notion of probabilistic graph, where edges and connections in a graph exist with some probability).

If we have a ground clause $(\text{connected}(a, b) \leftarrow \text{edge}(a, c), \text{connected}(c, b))$, we say that the clause is *true* relative to an interpretation if $|\text{edge}(a, c)| \wedge |\text{connected}(c, b)| \leq |\text{connected}(a, b)|$ in $[0, 1]$.

3 The Syntax Viewed Through Fibers

In this section we give a fibrational, or equivalently, indexed category based semantics to logic programs. In this section, we consider only the syntax of logic programs, prior to assigning any truth values, and so we essentially rephrase the fibrational semantics outlined in [9]. The reader can find missing definitions and explanations in [14,2,10].

We start by giving a structural interpretation to terms.

Definition 4. *Given $(\mathcal{T}, \mathcal{F})$ (before one adds the set $\mathcal{P}$ of predicate symbols) and given a category C with strictly associative finite products, a* pre-interpretation *of $(\mathcal{T}, \mathcal{F})$ in C is a function $\gamma : \mathcal{T} \to \mathtt{ob}(C)$ together with, for each function symbol f of sort $(\overline{T}, T)$ a map in C from $\gamma(T_1) \times \ldots \times \gamma(T_n)$ to $\gamma(T)$.*

One needs to show that such pre-interpretation exists and that it is unique. (Uniqueness of (pre)-interpretation is synonymous to its minimality in conventional terminology.) This was proved in [9] by constructing the category $C_{\mathcal{T},\mathcal{F}}$ with strictly associative finite products and the unique pre-interpretation $\| \ \|_{\mathcal{T},\mathcal{F}}$

of $(\mathcal{T}, \mathcal{F})$ in $C_{\mathcal{T},\mathcal{F}}$, as follows. The objects of $C_{\mathcal{T},\mathcal{F}}$ are finite sequences of elements of $\mathcal{T}$. An arrow from $\overline{T}$ to T is an equivalence class of terms of arity $T_1 \times \ldots \times T_n$ and type T, i.e, terms are factored out by renaming of variables.

Having interpreted terms, we continue with interpretation for formulae. For this, we need the notion of an indexed category with finite products.

Definition 5. *An* indexed category *over a small category C is a functor $p : C^{op} \to Cat$. An* indexed functor *from p to q is a natural transformation $\tau : p \Rightarrow q : C^{op} \to Cat$.*

Let FP_s be the category of small categories with finite products and functors that strictly preserve finite products.

Definition 6. *If a small category C has finite products, an indexed category $p : C^{op} \to Cat$ has* finite products *if $p : C^{op} \to Cat$ factors through FP_s, i.e, there is a functor $f : C^{op} \to FP_s$ such that $p = U \circ f$, where $U : FP_s \to Cat$ is inclusion.*

An indexed functor $h : p \Rightarrow q$ between indexed categories with finite products respects finite products *if each component does so.*

We say that p has *strictly associative* finite products if C and each $p(X)$ have strictly associative finite products.

We extend the definition of an interpretation of $(\mathcal{T}, \mathcal{F})$ in $C_{\mathcal{T},\mathcal{F}}$ to an interpretation of a language $\mathcal{L} = (\mathcal{T}, \mathcal{F}, \mathcal{P})$ in an indexed category with finite products over $C_{\mathcal{T},\mathcal{F}}$ as follows.

Definition 7. *An* interpretation of a sorted language $\mathcal{L} = (\mathcal{T}, \mathcal{F}, \mathcal{P})$ *in an indexed category $p : C_{\mathcal{T},\mathcal{F}} \to Cat$ with finite products is given by the pre-interpretation $\| \ \|_{\mathcal{T},\mathcal{F}}$: of $(\mathcal{T}, \mathcal{F})$ in $C_{\mathcal{T},\mathcal{F}}$, together with, for each sort $\overline{T} = T_1, \ldots T_n$, a function $\| \ \|_{\mathcal{P}(\overline{T})} : \mathcal{P}(\overline{T}) \to \mathtt{ob}(p(\|T_1\|_{\mathcal{T},\mathcal{F}} \times \ldots \times \|T_n\|_{\mathcal{T},\mathcal{F}}))$.*

Existence and uniqueness of such interpretation was proved in [9]. The free indexed category $p_{\mathcal{L}}$ with strictly associative finite products over $C_{\mathcal{T},\mathcal{F}}$, with an interpretation $\| \ \|_{\mathcal{L}}$ of $\mathcal{L}$ in $p_{\mathcal{L}}$ for a sorted language $\mathcal{L} = (\mathcal{T}, \mathcal{F}, \mathcal{P})$, is given as follows.

* For each $\overline{T} \in \mathtt{ob}(C_{\mathcal{T},\mathcal{F}})$, $p_{\mathcal{L}}(\overline{T})$ is the category with strictly associative finite products freely generated by $(\Phi_{\overline{T}}, \emptyset)$, where $\Phi_{\overline{T}}$ is the set of all triples $(\overline{U}, P, v)$ with $\overline{U} \in \mathrm{Sort}(\mathcal{T})$, a predicate symbol $P \in \mathcal{P}(\overline{U})$ and an arrow $v \in C_{\mathcal{T},\mathcal{F}}(\overline{T}, \overline{U})$. (The symbol $\emptyset$ in $(\Phi_{\overline{T}}, \emptyset)$ indicates that the logic programming arrows "$\leftarrow$" are not interpreted yet. Finite products of triples $(\overline{U}, P, v)$ give account to finite conjunctions.)

** For each $v \in C_{\mathcal{T},\mathcal{F}}(\overline{T}, \overline{U})$, we define the functor $p_{\mathcal{L}}(v) : p_{\mathcal{L}}(\overline{U}) \to p_{\mathcal{L}}(\overline{T})$ by specifying the value of $p_{\mathcal{L}}(v)(\overline{V}, P, s)$, with $s \in C_{\mathcal{T},\mathcal{F}}(\overline{U}, \overline{V})$, to be $(\overline{V}, P, s \circ v)$.

We can identify an object of $p_{\mathcal{L}}(\overline{T})$ with an equivalence class of finite sequences of atomic formulae with free variables of sort $\overline{T}$. We treat the finite sequence as a conjunction.

Definition 8. *Given a logic program Γ over the language $\mathcal{L}$, an* interpretation *of Γ in an indexed category p with strictly associative finite products is given by the following data:*

- *an interpretation $\| \ \|$ of $\mathcal{L}$ in p and*
- *for each sort $\mathcal{T}$, formula $\overline{\varphi}$ and atomic formula φ in $p(\overline{T})$, a function $\| \ \| : \Gamma_{\overline{T}}(\overline{\varphi}, \varphi) \to p(\overline{T})(\|\varphi_1\| \times \ldots \times \|\varphi_n\|, \|\varphi\|)$, where $\Gamma_{\overline{T}}(\overline{\varphi}, \varphi)$ is the family of clauses in Γ of the form $(\overline{T}, \overline{\varphi}, \varphi)$.*

The existence and uniqueness of such interpretation was proved in [9]. The unique interpretation was called p_Γ, and was essentially $p_\mathcal{L}$, but with added arrows that model the implication arrows "$\leftarrow$".

Example 6. In Example 4, categories $p_\Gamma(1)$, $p_\Gamma(D)$, $p_\Gamma(DD)$, $p_\Gamma(DDD)$, $p_\Gamma(NDDD)$ would be "fibers" generated by clauses of corresponding types.

In the many-valued setting that we will develop in the following sections, our attention will be on indexed category p for which each $p(\overline{T})$ is a preorder Ω with finite meets. In this case, the new "condition" in p_Γ amounts to the assertion that each clause $\varphi \leftarrow \overline{\varphi}$ is sent to an inequality $\|\varphi_1\| \wedge \ldots \wedge \|\varphi_n\| \leq \|\varphi\|$.

4 Ground Semantics, Fibrationally

We first show how the fibrational semantics fits into the framework of traditional declarative (ground) semantics.

We first choose a preorder Ω with finite meets in which to take values. By ground semantics for the underlying language $\mathcal{L}$ we mean the assignment, to each ground formula, of an element of Ω, respecting the structure of $\mathcal{L}$. This amounts to a finite product preserving functor from $p_\mathcal{L}(1)$ to Ω, where the latter is seen as a category with finite products. We extend it to the logic program Γ.

By previous discussions, $C_{\mathcal{T},\mathcal{F}}$ is the category with strictly associative finite products freely generated by $(\mathcal{T}, \mathcal{F})$. Let 1 be the terminal object of $C_{\mathcal{T},\mathcal{F}}$. So, for each $\overline{T} \in \mathtt{ob}(C_{\mathcal{T},\mathcal{F}})$, the homset $C_{\mathcal{T},\mathcal{F}}(1, \overline{T})$ is the set of ground terms of type $\overline{T}$. Moreover, $p_\mathcal{L}(1)$ is the category with strictly associative finite products freely generated by $(\Phi_I, \emptyset)$, with Φ_I being the set of all triples $(\overline{U}, P, v)$, where $v \in C_{\mathcal{T},\mathcal{F}}(1, \overline{U})$. Thus $p_\mathcal{L}(1)$ is the set of all ground formulae of the language $\mathcal{L}$ with finite meets, and it corresponds to the *Herbrand base*. An *interpretation* $| \ |$ of $\mathcal{L}$ in Ω is defined to be a finite meet preserving function from $p_\mathcal{L}(1)$ to Ω.

We now consider clauses. We do not simply assert that each clause is sent to an inequality in Ω, as that is not the practice in many-valued logic programming. We must allow unit clauses, i.e., clauses of the form $\varphi \leftarrow$, to be assigned values other than 1. We do this as follows.

Definition 9. *Given a many-valued annotation-free logic program Γ over the language $\mathcal{L}$, a* valuation *v of Γ in a preorder Ω with finite meets is an assignment to each unit clause $\varphi \leftarrow$ of Γ of an element $v(\varphi \leftarrow)$ of Ω.*

The notion of a valuation is often used in many-valued logic programming to describe a map from the elements of the Herbrand base to Ω. In our setting, the latter map would be redundant. Using Definition 9, we can interpret clauses directly, as follows.

Definition 10. *Given an annotation-free logic program Γ over the language $\mathcal{L}$, and a valuation v of Γ, a* ground interpretation *of Γ* with respect to the valuation v *in a preorder Ω with finite meets is an interpretation $|\ | : p_{\mathcal{L}}(1) \to \Omega$ of $\mathcal{L}$ such that for each clause in Γ of the form $\varphi \leftarrow \overline{\varphi}$, with $\overline{\varphi}$ non-empty, and each ground substitution $[g]$,*

$$|\varphi_1[g]| \wedge \ldots \wedge |\varphi_n[g]| \leq |\varphi[g]|,$$

and, for each unit clause $\varphi \leftarrow$ and ground substitution g, $v(\varphi \leftarrow) \leq |\varphi[g]|$.

Due to its inductive nature, this definition corresponds to the notion of the semantic operator (and its iterations) for many-valued logic programs; that is, the ground interpretation of a program is computed stepwise, starting with formulae which have received their valuation and then computing values for the rest of the formulae using the given data.

Example 7. If we fix $[0,1]$ to be the chosen preorder, then a valuation for the logic program GC from Example 1 can be given as follows.
$v(\text{connected}(x,x) \leftarrow) = 1,\ v(\text{edge}(a,b) \leftarrow) = 0.75,\ v(\text{edge}(b,c) \leftarrow) = 0.25$

The minimal ground interpretation would be given by
$|\text{connected}(a,a)| = 1,\ |\text{connected}(b,b)| = 1,\ |\text{connected}(c,c)| = 1;$
$\min(|\text{edge}(a,b)| = 0.75, |\text{connected}(b,b)| = 1) \ \leq\ |\text{connected}(a,b)| = 0.75,$
$\min(|\text{edge}(b,c)| = 0.25, |\text{connected}(c,c)| = 1) \ \leq\ |\text{connected}(b,c)| = 0.25,$
$\min(|\text{edge}(a,b)| = 0.75, |\text{connected}(b,c)| = 0.25) \ \leq\ |\text{connected}(a,c)| = 0.25,$
$|\text{edge}(a,b)| = 0.75,\ |\text{edge}(b,c)| = 0.25$

There is a standard way of defining a minimal model for many-valued logic programs, described, for example, in [7,4]. We can emulate this in our own terms by defining an ordering on a set of all the *ground interpretations* as follows. Let $|\ |_1$ and $|\ |_2$ be ground interpretations with respect to a valuation v for a logic program Γ over the language $\mathcal{L}$. Then we say that $|\ |_1 \leq |\ |_2$ if $|\varphi[g]|_1 \leq |\varphi[g]|_2$ for every ground substitution of every formula φ in Γ. The set of all ground interpretations forms a preorder M with objects the ground interpretations and arrows given by $\leq$ defined as above. We define the *ground model* of an annotation-free logic program Γ to be the least element of M.

One needs to be careful in regard to the ground models as the following examples illustrate.

Example 8. Consider a logic program of the form $p(a) \leftarrow$, $p(x) \leftarrow q(x)$, $q(a) \leftarrow$, with valuation v in $[0,1]$ given by $v(p(a) \leftarrow) = 0.3$; $v(q(a) \leftarrow) = 0.7$.

By Definition 10, in any ground interpretation, $0.3 \leq |p(a)|$, $0.7 \leq |q(a)|$, and also $|q(a)| \leq |p(a)|$. Thus, $0.7 \leq |p(a)|$ in any ground interpretation. So, there is a one-step proof that $0.3 \leq |p(a)|$ and a two-step proof that $0.7 \leq |p(a)|$.

This situation evidently can be extended to logic programs involving proofs of indefinite length, so needs to be taken seriously when giving SLD-resolution, in particular in determining the ground model.

Example 9. Consider the logic program of Example 8 with valuation in $[0,1] \times [0,1]$ given by $v(p(a) \leftarrow) = (0, 0.5)$; $v(q(a) \leftarrow) = (0.5, 0)$. Then, in any ground interpretation, $(0.5, 0) \leq |p(a)|$ and $(0, 0.5) \leq |p(a)|$, so $(0.5, 0.5) \leq |p(a)|$, but there is no computation that shows this directly. This will lead us to requiring finite joins in Ω in Section 6. Variants of this example exist in Kleene's logics and logics which generalise Kleene's logics, [6,7].

5 Fibrational Many-Valued Semantics

The fibrational semantics will provide us with non-ground interpretations for logic programs. In Theorem 1 we relate ground and fibrational semantics.

Let $\mathcal{C}$ be a small category and $\mathcal{D}$ have all products, and let 1 be a terminal object of $\mathcal{C}$. The diagonal functor $\Delta : \mathcal{D} \to \mathcal{D}^{\mathcal{C}^{\mathrm{op}}}$ has a right adjoint given by sending $F \in \mathcal{D}^{\mathcal{C}^{\mathrm{op}}}$ to $F(1)$. I.e., a right adjoint to the diagonal is given by evaluation at 1, and we will denote the right adjoint by $ev_1 : \mathcal{D}^{\mathcal{C}^{\mathrm{op}}} \to \mathcal{D}$.

Proposition 1. *The functor $ev_1 : \mathcal{D}^{\mathcal{C}^{op}} \to \mathcal{D}$ has a right adjoint $R : \mathcal{D} \to \mathcal{D}^{\mathcal{C}^{op}}$, given by $R(D)(C) = D^{\mathcal{C}(1,C)}$, for each $D \in \mathcal{D}$ and each $C \in \mathcal{C}^{op}$.*

Corollary 1. *The functor $ev_1 : \mathtt{FP}_s^{C_{\mathcal{T},\mathcal{F}}} \to \mathtt{FP}_s$ has a right adjoint given by $R(\Omega) = \Omega^{C_{\mathcal{T},\mathcal{F}}(1,-)}$.*

Recall that in Section 4, we studied maps of the form $p_{\mathcal{L}}(1) \to \Omega$ in $\mathtt{FP}_s$. By Corollary 1, they are equivalent to natural transformation $p_{\mathcal{L}} \to \Omega^{C_{\mathcal{T},\mathcal{F}}(1,-)}$. So, consider a natural transformation $\psi : p_{\mathcal{L}} \to \Omega^{C_{\mathcal{T},\mathcal{F}}(1,-)}$. This is equivalent to giving, for each $\overline{T}$ and each ground term $\overline{t}$ of sort $\overline{T}$, a finite meet preserving function $|\ | : p_{\mathcal{L}}(1) \to \Omega$ natural in $\overline{T}$.

Since Ω is a preorder with finite meets, $\Omega^{C_{\mathcal{T},\mathcal{F}}(1,\overline{T})}$ has a preorder structure with finite meet given pointwise. We use that fact in our definition of fibrational interpretation, which by the above discussion will be equivalent to Definition 10.

Definition 11. *Given an annotation-free logic program Γ over the language $\mathcal{L}$, a* fibrational interpretation*, or* f-interpretation*, with respect to the valuation v of Γ in Ω is given by an interpretation $\|\ \|$ of $\mathcal{L}$ in $\Omega^{C_{\mathcal{T},\mathcal{F}}(1,-)}$, such that:*

- *For each unit clause $\varphi \leftarrow$ in Γ, $v(\varphi \leftarrow) \leq \|\varphi\|$;*
- *For each clause in Γ of the form $\varphi \leftarrow \overline{\varphi}$, where $\overline{\varphi}$ is non-empty,*

$$\|\varphi_1\| \wedge \ldots \wedge \|\varphi_n\| \leq \|\varphi\|.$$

Theorem 1. *Given an annotation-free logic program Γ over the language $\mathcal{L}$, a preorder Ω, and a valuation v of Γ in Ω, to give an f-interpretation with respect to v is equivalent to giving a ground interpretation of Γ with respect to v.*

Proof. This follows from the adjointness of Corollary 1 and the definition of interpretation and valuation.

Example 10. We take the valuation of the program GC from Example 7. The minimal f-interpretation generated by the valuation is:
$\|\text{connected}(x,x)\| = 1$, $\|\text{edge}(a,b)\| = 0.75$, $\|\text{edge}(b,c)\| = 0.25$,
$\min(\|\text{edge}(x,z)\|, \|\text{connected}(z,y)\|) \leq \|\text{connected}(x,y)\|$.

The last line subsumes all the possible substitutions. Notably, ground substitutions agree with the ground interpretation for GC from Example 7.

Given a logic program Γ, we will call the least f-interpretation of Γ an *f-model* for Γ. It is the least element in the preorder of all f-interpretations of Γ, similarly to Section 4.

6 SLD-Resolution

Motivated by our fibrational semantics, we give a definition of the SLD-resolution for annotation-free logic programs. The idea is as follows. The syntax of annotation-free logic programs is exactly the same as that of conventional logic programs. So we can first do SLD-resolution for an annotation-free logic program qua conventional logic program which is expressible in terms of fibrational semantics and is sound and complete with respect to fibrational semantics; [9]. Now we introduce valuations. Given a refutation tree, we consider the leaves. These amount to unit clauses, so have valuation. We then proceed in the backward direction from the leaves to the root of the refutation tree to generate a minimal value for the substituted goal. Note that the leaves are not necessarily ground, and hence fibrational rather than ground approach is appropriate.

We restrict the choice of Ω by requiring Ω to have all, not only finite, meets. The existence of all meets in Ω implies the existence of all joins. A delicate analysis allows us to restrict to finite joins in addition to finite meets. As Example 9 indicates, we need some such assumption in order to justify the existence of ground models and f-models for annotation-free logic programs.

We start with a definition of SLD-resolution in terms of state transition machines. See also [9], where mgus were characterised as *pullbacks*. We will call $[s_1, s_2]$ an mgu of atomic formulae A and B with terms modelled by arrows u respectively v in $C_{\mathcal{T},\mathcal{F}}$, if $[s_1, s_2]$ is an mgu of u and v.

Definition 12. *Given an annotation-free logic program Γ in $\mathcal{L}$, the* state transition machine *M_Γ associated to Γ is the directed graph (N, E) defined as follows. N is the set of all formulae in $\mathcal{L}$. An edge with source $\varphi = \varphi_1 \times \ldots \times \varphi_n$ is a triple $(l, \rho, (s_1, s_2))$, where $l : H \leftarrow B$ is a clause in Γ, $\rho = \pi_i : \overline{\varphi} \to \varphi_i$ is the projection to φ_i, and (s_1, s_2) is an mgu for φ_i and H. The target of $(l, \rho, (s_1, s_2))$ is $\varphi_1[s_1, s_2] \times \ldots \times \varphi_{i-1}[s_1, s_2] \times B[s_1, s_2] \times \varphi_{i+1}[s_1, s_2] \times \ldots \times \varphi_n[s_1, s_2]$.*

Definition 13. *Given a logic program Γ and a goal G in $\mathcal{L}$, a* computation *in M_Γ with goal G is a directed path T in M_Γ starting at G, in particular, if the*

endpoint is a terminal 1 *in some fibre of* $p_{\mathcal{L}}$*, then it is said to be a* successful computation *or* refutation. *Finally, if*

$$G = \overline{\varphi} \xrightarrow{(l_1,\rho_1,(s_1^1,s_2^1))} \overline{\varphi} \xrightarrow{(l_2,\rho_2,(s_1^2,s_2^2))} \dots \xrightarrow{(l_{m-1},\rho_{m-1},(s_1^{m-1},s_2^{m-1}))} \overline{\varphi}$$

is a computation, $(s_1^1, s_2^1), (s_1^2, s_2^2), \dots, (s_1^{m-1}, s_2^{m-1})$ *is defined to be its* answer.

The SLD-refutation is sound and complete with respect to the (two-valued) fibrational semantics, that is, the following theorem holds:

Theorem 2. *[9] Let* Γ *be a logic program in* $\mathcal{L}$*. Substitution* $s : \overline{U} \to \overline{T}$ *is the answer of a refutation in* M_Γ *with goal* G *of sort* $\overline{T}$ *if and only if there is an arrow* $m : 1 \to G[s]$ *in the fibre* $p_\Gamma(\overline{U})$*.*

Next we introduce a valuation into a mechanism of refutation to the annotation-free logic programs and give the inductive definition of a tree computing a value for the goal G as follows.

Definition 14. *Let* M_Γ *be the state transition machine associated to a logic program* Γ *and a goal* G *as in Definition 12. Let* T *be a directed path in* M_Γ *such that* T *performs a refutation for a formula* G *in* $\mathcal{L}$ *with the computed answer* s*, and let* v *be a valuation of* Γ*. A* computation of a value *for* G *is a directed path* T^{op} *starting at* 1 *and ending at* $\|G[s]\|$ *in* Ω*, such that the following holds:*

1. *Whenever there is an edge* $(l, \rho, (s_1, s_2))$ *from* $\varphi_1[s_1, s_2] \times \dots \times \varphi_{i-1}[s_1, s_2] \times \varphi_{i+1}[s_1, s_2] \times, \dots, \times \varphi_n[s_1, s_2]$ *to* $\overline{\varphi} = \varphi_1 \times \dots \times \varphi_i \times \dots \times \varphi_n$*, as described in Definition 12, with* $\rho = \pi_i$ *and* l *of the form* $H \leftarrow$*, then we use the valuation* v *of* H *and substitute* φ_i *in* $\overline{\varphi}$ *by* $v(H)$*.*
2. *For every edge* $(l, \rho, (s_1, s_2))$ *from* $\varphi_1[s_1, s_2] \times \dots \times \varphi_{i-1}[s_1, s_2] \times B_1[s_1, s_2], \dots, B_k[s_1, s_2] \times \varphi_{i+1}[s_1, s_2] \times, \dots, \times \varphi_n[s_1, s_2]$ *to* $\varphi = \varphi_1 \times \dots \times \varphi_n$*, with* $\rho = \pi_i$ *and* l *of the form* $H \leftarrow B_1, \dots B_k$*, we use* $v(B_1) \wedge \dots \wedge v(B_k)$ *to transform the node* $\overline{\varphi}$ *into* $\varphi_1 \times \dots \times \varphi_{i-1} \wedge (v(B_1) \wedge \dots \wedge v(B_k)) \wedge \varphi_{i+1} \times \dots \times \varphi_n$*.*

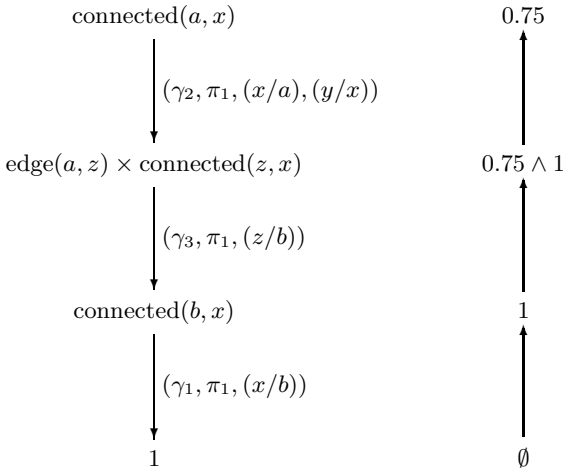
It is easy to see that for every such *computation of a value* for $G = \varphi'_1 \times \dots \times \varphi'_m$, the endpont of T^{op} will be $\|G[s]\| = \bigwedge(v(B'_1[s])) \wedge \dots \wedge \bigwedge(v(B'_m[s]))$, where each $\bigwedge(v(B'_j[s]))$ performs the value of the goal atom $\varphi'_j[s]$.

Definition 15. *Let* Γ *be an annotation-free logic program interpreted in a pre-order* Ω *with the least element* 0*. Let* G *be a goal in* Γ*. We say that* $\omega \in \Omega$ *is a* computed value *for* G *if one of the following conditions holds:*

- *There is a refutation for* G *with answer* s *and the algorithm of* computation of a value *outputs* $\|G[s]\| = \omega$*;*
- *There is no refutation for* G *and we put* $\omega = 0$*.*

Example 11. Consider the logic program GC from Example 1 and the goal G of the form (connected(a, x)). The leftmost tree T performs a refutation for G with the answer $s = (x/b)$. The rightmost tree T^{op} shows how the value for $G[s]$

is computed by the algorithm of computation of a value. We use the valuation v from Example 10.



Thus, the goal (connected$(a, x)[x/b]$) receives the computed value 0.75. Note that this agrees with the minimal ground interpretation of GC from Example 7 and f-model of GC from Example 10.

The algorithm of *computation of a value* for $G[s]$ is sound and complete with respect to both ground model and f-model of a logic program.

Theorem 3 (Soundness relative to fibrational semantics). *Let Γ be a logic program and G be a goal formula, such that there is a tree T in the state transition machine M_Γ and T performs refutation for $\{\Gamma \cup G\}$ with the computed answer s. Then the following holds. If the algorithm of* computation of a value *outputs the value ω for $G[s]$, then in the f-model of Γ, $\|G[s]\| \geq \omega$.*

Proof. We use Theorem 2; the rest of the proof proceeds by induction on the length of the tree $T \in M_\Gamma$.

Theorem 4 (Completeness relative to fibrational semantics). *If $\|G[s]\| = \omega$ is in the f-model of Γ, then there exists a finite set of trees $T_1, \ldots, T_n$ which compute the substitution s as answer, such that ω is the supremum of the computed values for $G[s]$ in $T_1, \ldots, T_n$.*

Proof. We use Theorem 2; then we proceed by induction on complexity of clauses interpreted by the ground model of Γ. Finite joins are required in order to account for cases such as Examples 8, 9. Only finite joins are needed as each valuation v only makes finitely many assignments. (Note that as we have assumed the existence of all meets in Ω, it follows that Ω also has finite joins.)

In practice, one needs to use the conventional algorithm of backtracking to compute all the values. Annotation-free logic programs can have infinitely long computations and infinitely long trees T in M_Γ, as in Example 12:

Example 12. The following logic program may have infinitely long refutations for the goal ($\leftarrow p(x)$): $q(x) \leftarrow$, $p(x) \leftarrow q(x)$, $q(x) \leftarrow p(x)$.

But the number of unit clauses in any logic program is finite, and so is the number of values assigned to them. This is why, refutations for annotation-free logic programs will always have finitely many computed values. In our example, the only possible computed value for $p(x)$ will be the value $v(q(x) \leftarrow)$.

We now show that traditional-style soundness and completeness of the SLD-resolution relative to the ground semantics can be obtained as a corollary of Theorems 1, 3, 4. We make use of Theorem 1 and use $|\ |$ instead of $\|\ \|$ when talking about interpretations for ground atoms.

In conventional logic programming [12], one speaks of the *success set* of a program Γ. That is the set of all ground atoms for which refutation exists. We cannot directly use that definition here because of the presence of non-trivial values. But, to give the success set of a conventional logic program is equivalent to giving function from $p_{\mathcal{L}}(1)$ to $\{0, 1\}$, satisfying a success condition. We could call that the success map corresponding to the success set, cf. [5]. So we generalise the success map as follows.

Definition 16. *Given an annotation-free logic program Γ over $\mathcal{L}$, a preorder Ω with meets, and a valuation v of Γ in Ω, the* success map *of Γ is the map $|\ | : p_{\mathcal{L}}(1) \to \Omega$ such that for each ground instance $\varphi[g]$ of a formula φ, $|\varphi[g]|$ is the supremum of all computed values ω of $\varphi[g]$.*

The soundness and completeness of the algorithm of *computation of a value* relative to the ground model of a given program can now be stated as follows.

Corollary 2 (Soundness and completeness relative to ground semantics). *Let Γ be a many-valued annotation-free logic program. The success map of Γ is equal to its ground model.*

7 Conclusions and further work

We have given ground and fibrational semantics to many-valued logic programming. We have proved theorems showing the exact relationship between the two kinds of semantics. This gave theoretical justification of the appropriateness of the fibrational (non-ground) approach to logic programming semantics. Fibrational semantics easily relates to existing resolution procedures [9] and gives rise to novel proof search algorithms. In particular, we have developed the novel algorithm of SLD-resolution for annotation-free many-valued logic programs. We proved that this algorithm is sound and complete relative to the fibrational semantics and showed that soundness and completeness of the algorithm relative to ground semantics can be obtained as a corollary of that.

Related work. Comparing with other kinds of many-valued resolution algorithms [3,16], the algorithm we have described turns out to be the ideal compromise between expressiveness and efficiency. Unlike [16], we do not impose

syntactical restrictions on the shape and groundness of clauses and goals; and instead allow any first-order definite logic program to be processed. E.g., programs as in Example 12 would not be allowed in the setting of [16]. So, fibrational approach gives a clear advantage of expressiveness.

The algorithm of [3] is not restricting the syntax of logic programs, but it is very complex and in general non-terminating. In [3] one has to go through 5 consecutive stages of building a forest the trees of which would present all possible branches of refutation. Our algorithm avoids this by using conventional method of backtracking to find all the possible values and substitutions.

It is remarkable that both [3] and [16] use ground semantics in order to prove soundness and completeness of the algorithms. And this adds complications. Thus, proofs of soundness and completeness of the resolution in [3] exclude the non-terminating cases like the one in Example 12. So, it would seem that the shift towards non-ground, fibrational approach to resolution simplifies the algorithm as well as makes proofs of soundness and completeness easier.

Further work may involve extensions of the fibrational semantics to other types of many-valued logic programs: implication-based and annotated (signed).

We intentionally analysed only the simplest type of logic programs, that is, definite programs which allow only conjunctions in clause bodies. One can adapt existing categorical interpretations of other connectives [11] to the setting.

We also hope that the result relating ground and fibrational semantics will open new horizons for the structural characterisation of other types of non-classical logic programs (such as (e.g.) multimodal, non-monotonic) whose declarative semantics depends on truth assignments.

References

1. M. Baaz, C. G. Fermuller, and G. Sazler. Automated deduction for many-valued logics. In A. Robinson and A. Voronkov, editors, *Handbook of Automated Reasoning*, volume 2, pages 1355 – 1402. Elsevier, 2001.
2. M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice-Hall, 1990.
3. C. Damásio and M. Medina, J. Ojeda-Aciego. A tabulation procedure for first-order residuated logic programs. In *Proc.IPMU-06*, 2006.
4. C. V. Damásio and L. M. Pereira. Sorted monotonic logic programs and their embeddings. In *Proc. IPMU-04*, pages 807–814, 2004.
5. S. E. Finkelstein, P. Freyd, and J. Lipton. Logic programming in tau categories. In *CSL'94*, volume 933 of *LNCS*. Springer, 1995.
6. M. Fitting. A Kripke/Kleene semantics for logic programs. *J. of logic programming*, 2:295–312, 1985.
7. M. Fitting. Fixpoint semantics for logic programming — a survey. *TCS*, 278(1-2):25–51, 2002.
8. M. Kifer and V. S. Subrahmanian. Theory of generalized annotated logic programming and its applications. *J. of logic programming*, 12:335–367, 1991.
9. Y. Kinoshita and A. J. Power. A fibrational semantics for logic programs. In *Proc. 5th Int. Workshop on Extensions of Logic Programming*, volume 1050 of *LNAI*, Leipzig, Germany, 1996. Springer.

10. E. Komendantskaya and J. Power. Fibrational semantics for many-valued logic programs. Tech. Report N 00295027, `http://hal.inria.fr/inria-00295027/en/`, INRIA, 2008.
11. J. Lambek and P. Scott. *Higher Order Categorical Logic*. Cambridge University Press, 1986.
12. J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.
13. J. J. Lu, N. V. Murray, and E. Rosenthal. Deduction and search strategies for regular multiple-valued logics. *J. of Multiple-valued logic and soft computing*, 11:375–406, 2005.
14. S. MacLane. *Categories for the working mathematician*. Springer-Verlag, Berlin, 1971.
15. A. J. Power and L. Sterling. A notion of a map between logic programs. In *Logic Programming, Proc. 7th Int. Conf.*, pages 390–404. MIT Press, 1990.
16. U. Straccia. A top-down query answering procedure for normal logic programs under the any-world assumption. In *Proc. ECSQARU-07*, number 4724 in LNCS, pages 115–127. Springer Verlag, 2007.
17. M. van Emden. Quantitative deduction and fixpoint theory. *J. Logic Programming*, 3:37–53, 1986.